Optimization Methods

For Deep Learning

Dr. Bethany Lusch
Assistant Computer Scientist
Argonne Leadership Computing Facility
blusch@anl.gov

August 7, 2020
ATPESC

What is Optimization?

“objective” or “loss” function

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & f(x) \\ \text{subject to} \quad & \ldots\text{constraints}\ldots \end{aligned}$$

Underneath most machine learning problems is an optimization problem

Example: Minimize prediction error

Typical Deep Learning Formulation

mean squared error, averaging over the examples $x^{(1)}, x^{(2)}, \ldots, x^{(n)}$

$$\min_{\boldsymbol{\theta}} \frac{1}{n} \sum_{i=1}^{n} [h(x^{(i)}; \boldsymbol{\theta}) - y^{(i)}]^2$$

Correct label for each example

“predictor” function: the neural network
Where $\boldsymbol{\theta}$ are trainable parameters

Recall: h has that special layered form, such as:

$$h(x; \boldsymbol{\theta}) = \sigma(W^{[2]}\sigma(W^{[1]}x + b^{[1]}) + b^{[2]})$$

Types of Optimization

Roughly…

Desperation

- Linear
- Quadratic
- Convex

More time on formulating problem to fit these categories

- Have 2nd derivs
- Have gradients
- General

More time on optimization algorithm

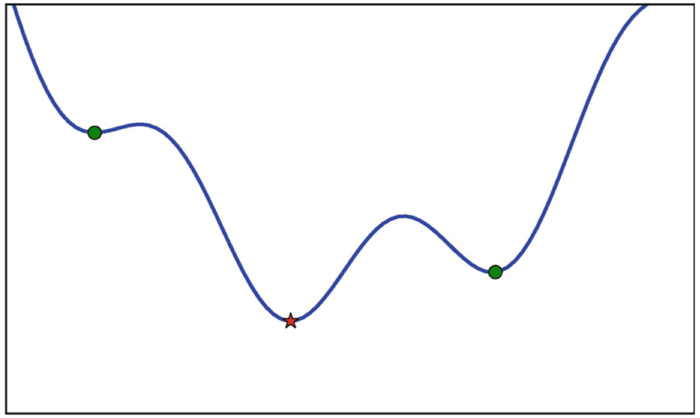
Differentiable Optimization

Desperation

- Linear
- Quadratic
- Convex
- Have 2nd derivs
- Have gradients
- General

Deep learning usually here

- Objective function non-convex
- So local minima problematic



- Technically have 2nd derivatives, but too expensive

Differentiable Optimization

Desperation

- Linear
- Quadratic
- Convex
- Have 2nd derivs
- Have gradients
- General

Deep learning usually here

- So typically use stochastic gradient descent
- In the optimization zone of: “our problem doesn’t have great properties, so spend time on the difficult optimization”

Choosing form of neural network

- Details of $h(x;\boldsymbol{\theta})$, i.e. $h(x;\boldsymbol{\theta}) = \sigma(W^{[2]}\sigma\left(W^{[1]}x + b^{[1]}\right) + b^{[2]})$

- Choosing "neural architecture" or "function family"

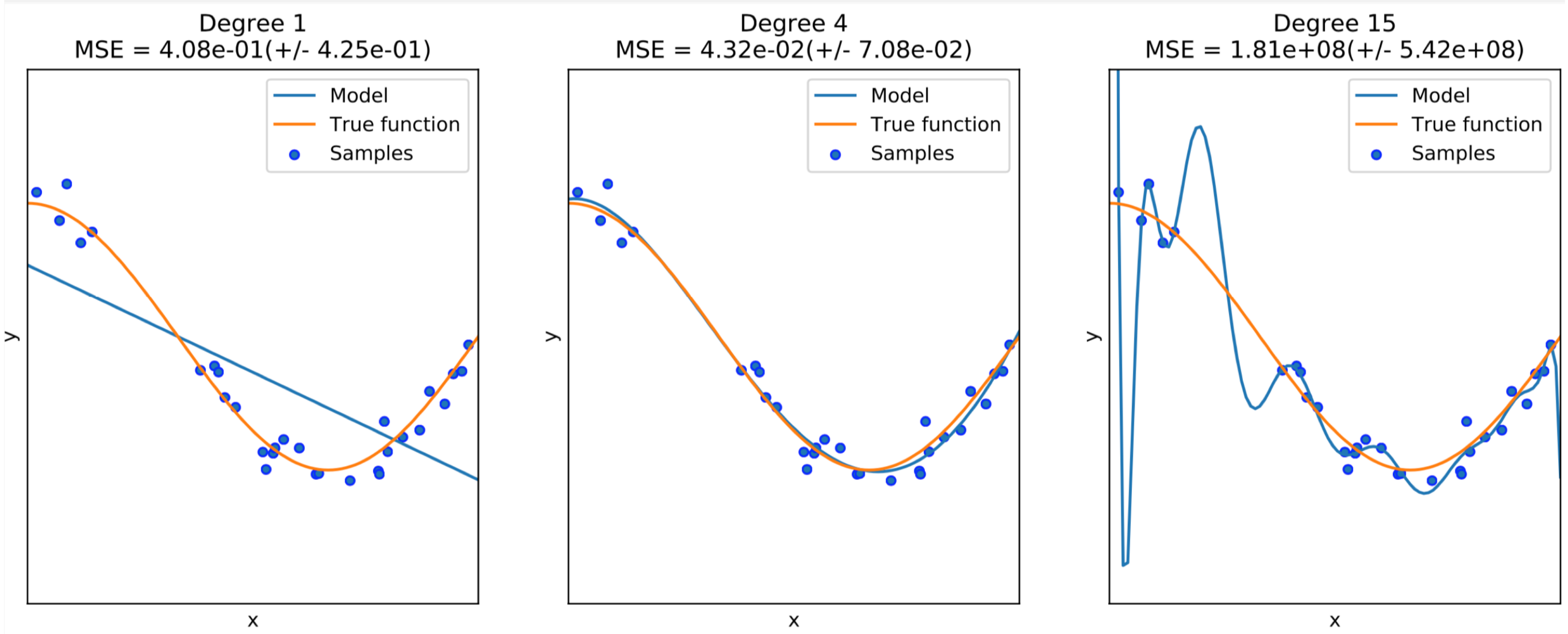
Pros of deep learning:
- Universal approximation theorem: can approximate "any" function arbitrarily well
- Hierarchical structure saves parameters

Good parameters might exist, but do you find them?

Cons of deep learning:
- Non-convex, so can be hard to find best parameters
- Can be overly flexible/complicated

Bias vs. Variance

underfitting | balanced | overfitting



High bias | Low bias, low variance | High variance

A major theme of machine learning!

Pictures from Kyle Felker, produced from code in scikit-learn documentation

To Check for Overfitting vs. Underfitting

"test" data

Rule #1: MUST hold out some data and check error *at very end*

Common:

- Randomly split data 70% training, 20% validation, 10% test
- Use training data to fit parameters of network
- Use validation data to compare options (like learning rate)
- Report test error at **end of project**

If you peek, not really reporting generalization error!

Choosing Hyperparameters

- Ex: learning rate, batch size, number of layers
- More at “Hyper-parameter Optimization” talk
- Common to try variety and choose “best” combination
  - Typically: lowest **validation** error in fixed number of epochs
  - Or fixed time…
  - If targeting particular error, could explore best time-to-solution

DO NOT consult your test error!!

To Check for Overfitting vs. Underfitting

Monitor training and validation error…

If training error too high → underfitting

If training error << validation error → overfitting

Extrapolation

mathematical sense of word

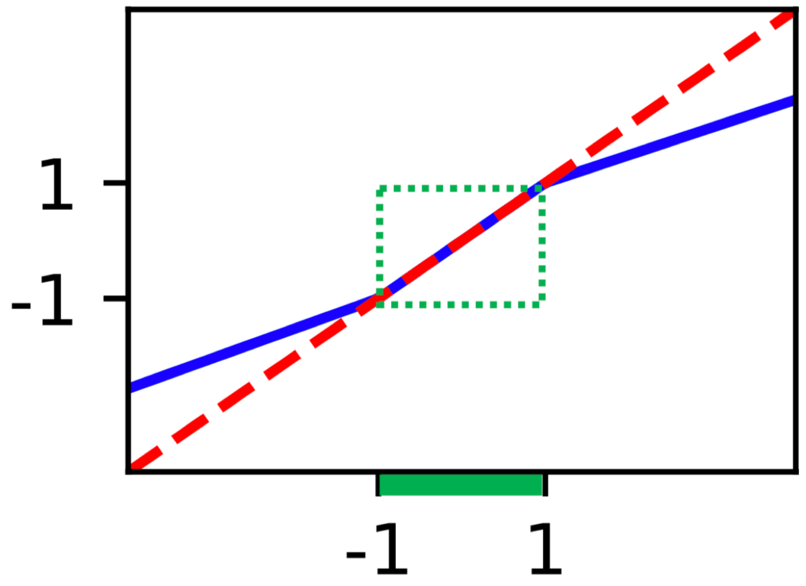
Rule #2: DO NOT extrapolate to inputs outside literal training domain

Cautionary example: Learn f(x) = x, for 1-D x

Noiseless training data on [-1, 1]

Trained tiny 6-parameter network, can write down perfect weights

Excellent val. error in [-1, 1] does not lead to extrapolation ability outside [-1, 1]

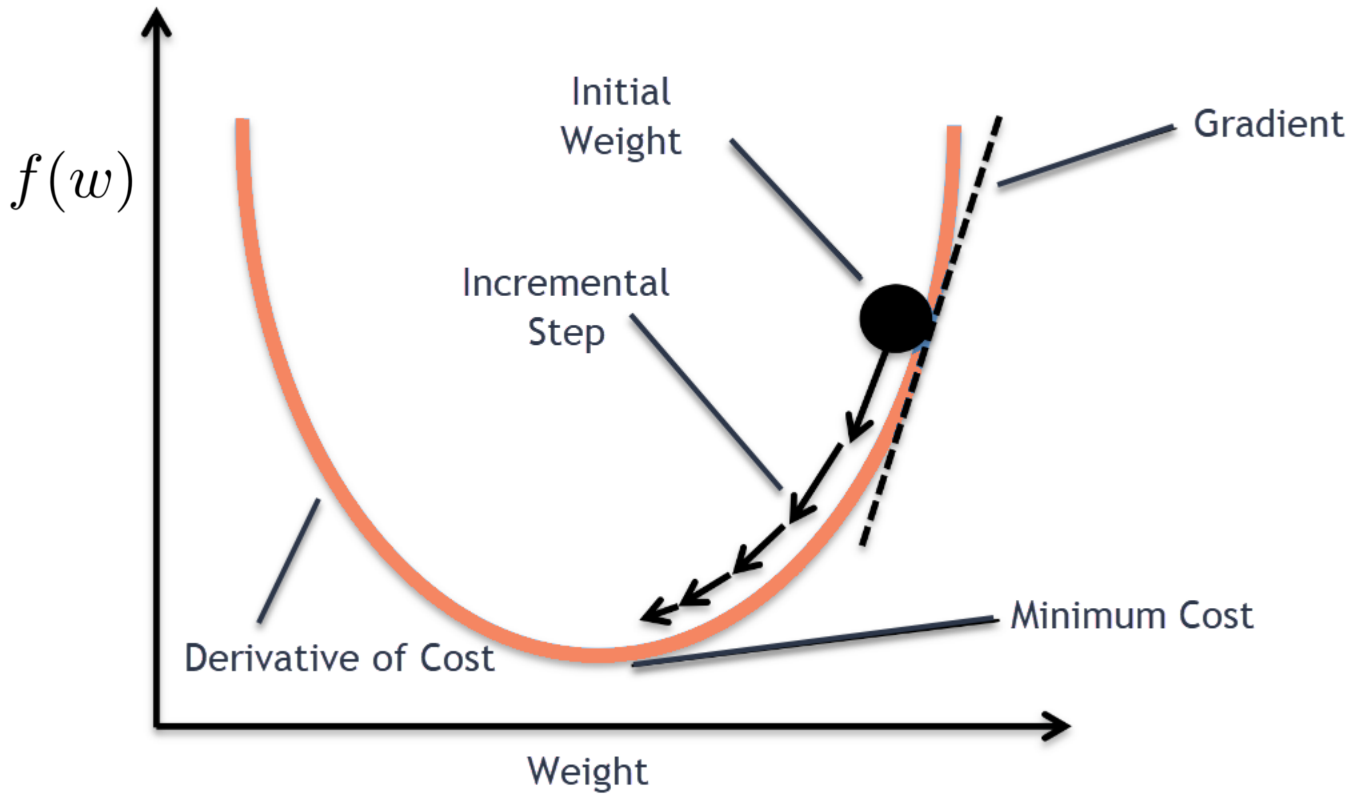


https://arxiv.org/abs/1911.02710

Gradient Descent

$$\underset{w}{\text{minimize}} \quad f(w)$$

$$w_{k+1} \leftarrow w_k - \alpha_k \nabla f(w_k)$$



Picture source: Divakar Kapil in “Stochastic vs Batch Gradient Descent”

Types of Gradient Descent

(in the context of summing a loss over examples)

$$w_{k+1} \leftarrow w_k - \frac{\alpha_k}{n} \sum_{i=1}^{n} \nabla f_{ik}(w_k)$$

Batch GD: use all examples every step

$$w_{k+1} \leftarrow w_k - \alpha_k \nabla f_{ik}(w_k)$$

Stochastic GD: use one example per step

$$w_{k+1} \leftarrow w_k - \frac{\alpha_k}{|S_k|} \sum_{i \in S_k} \nabla f_{ik}(w_k)$$

Mini-batch GD: use a subset each step

One epoch: use each example once

Types of Gradient Descent

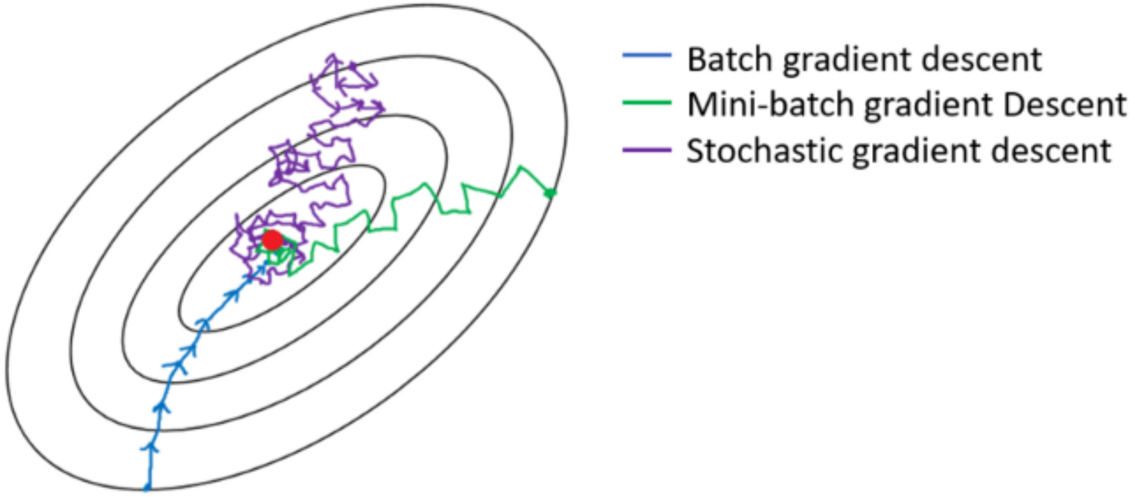
Batch GD: use all examples every step — Each step is accurate but expensive

Stochastic GD: use one example per step — Each step is noisy but fast

Mini-batch GD: use a subset each step — Happy medium?

Very common in deep learning, but often call it SGD

Even better: shuffle data between epochs so mini-batches change

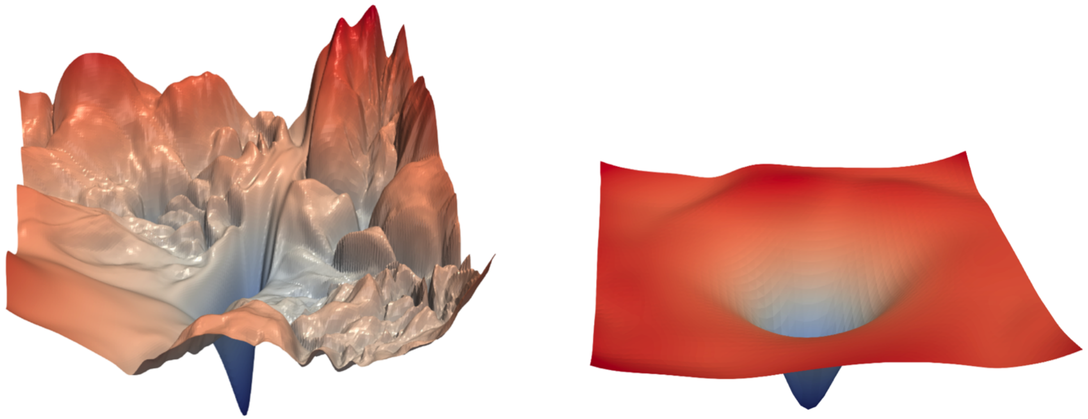


https://towardsdatascience.com/gradient-descent-algorithm-and-its-variants-10f652806a3

Learning Rate

$$\underset{w}{\text{minimize}} \quad f(w)$$

$$w_{k+1} \leftarrow w_k - \alpha_k \nabla f(w_k)$$



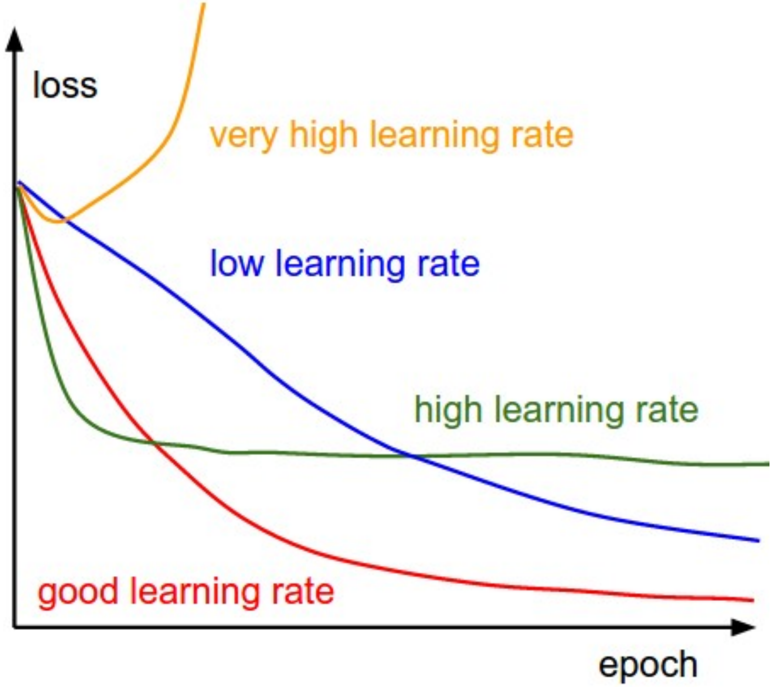
(a) without skip connections

(b) with skip connections

Li, et al. "Visualizing the Loss Landscape of Neural Nets" NeurIPS 2018

Learning rate is "step size"

- Too big: overshoot
- Too small: very slow
- (But might want to escape local minima)



http://cs231n.github.io/neural-networks-3/

Batch Size

$$w_{k+1} \leftarrow w_k - \frac{\alpha_k}{|S_k|} \sum_{i \in S_k} \nabla f_{ik}(w_k)$$

Mini-batch GD: use a subset each step

batch size

Choosing a batch size:

- Time per epoch
  - large often fast due to vectorization
- But accuracy!
  - Too small can be noisy steps
  - Too big can be get stuck in local minima

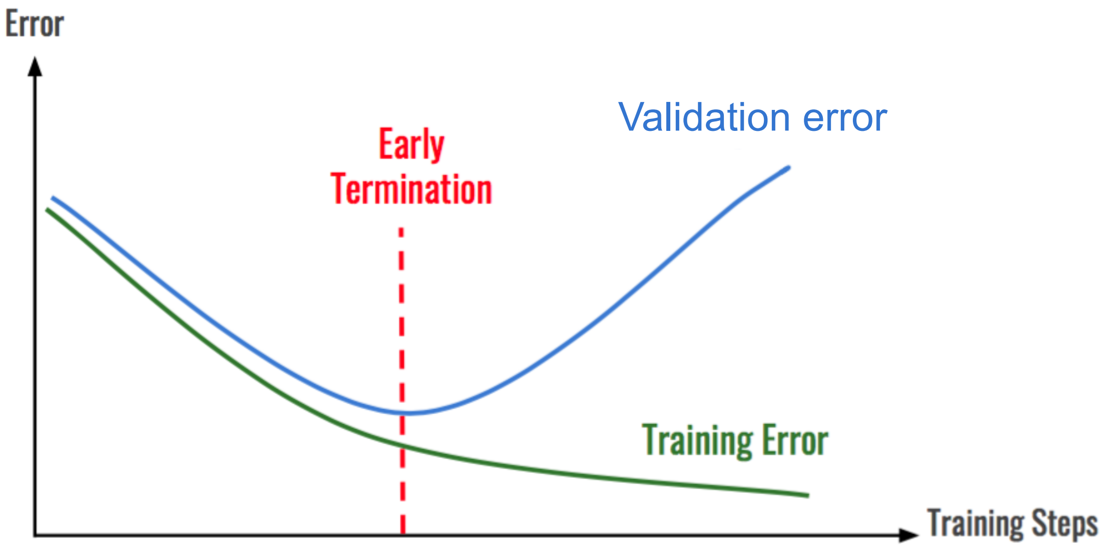
Argonne NATIONAL LABORATORY

Convergence

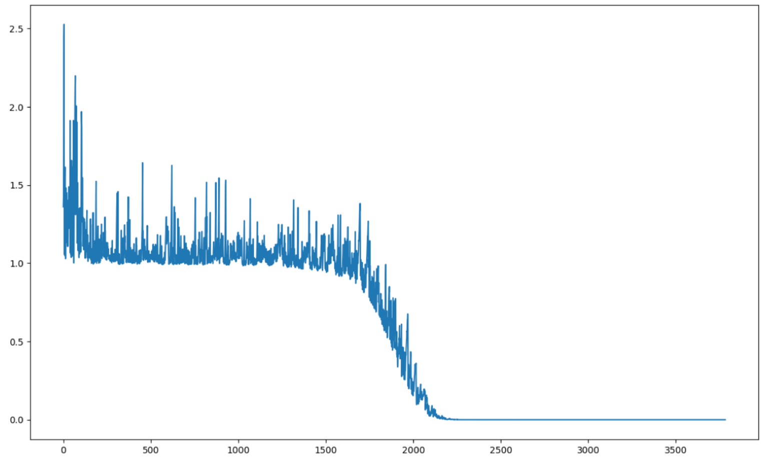
Monitor training & validation error

If validation error plateaued (or getting worse!) →

- Often "early stopping" (save best so far)
- Or tweak learning rate
- But might want to wait: could jump into different local minimum



https://hackernoon.com/memorizing-is-not-learning-6-tricks-to-prevent-overfitting-in-machine-learning-820b091dc42



https://stats.stackexchange.com/questions/257843/constant-error-during-training

Argonne NATIONAL LABORATORY

Variant: Adam Optimizer

Popular improvement on GD: Adam optimizer

- Separate learning rate for each weight
- Momentum: uses moving average of the gradient
- Also incorporates squared gradients

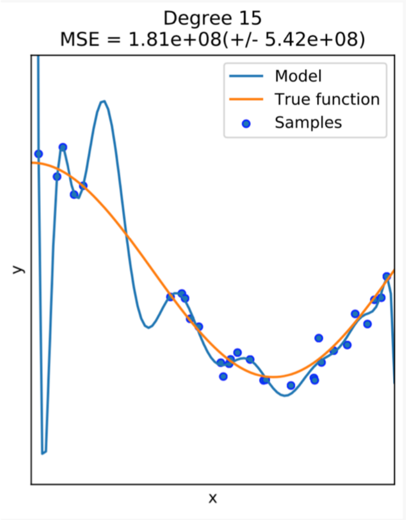
Cool exploration/visualization of momentum: https://distill.pub/2017/momentum/

(For those familiar: Adam combines the best properties of AdaGrad, momentum, and RMSProp)

Regularization

- Common way to avoid overfitting: regularization
- Most common: L2 regularization

$$\underset{w}{\text{minimize}} \quad \underbrace{\frac{1}{n}\sum_{i=1}^{n}\left(h(x_i;w)-y_i\right)^2}_{\text{error}} + \overset{\text{balance}}{\lambda}\underbrace{\|w\|_2^2}_{\text{regularization}}$$

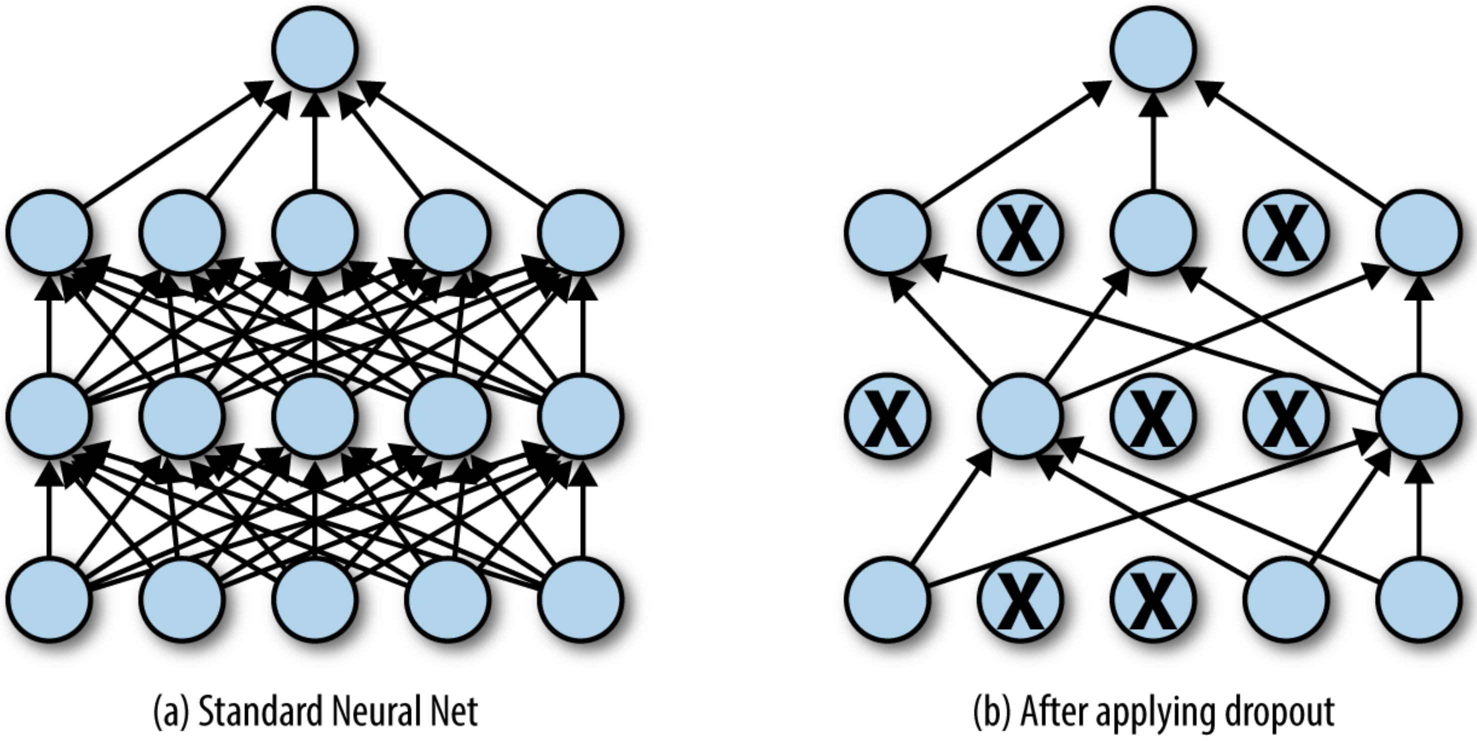


Roughly: big coefficients/weights correspond to large variation

Dropout

Probability keep node = p

- Apply during training time only
- Can define layer-by- layer
- Scale the surviving activations by 1/p
- Network has to be "resilient"



TensorFlow for Deep Learning by Bharath Ramsundar; Reza Bosagh Zadeh Figure 4-8

Adapted from Kyle Felker's slide

Summary

- Deep learning is an optimization problem
- Choices affect
  - Can the neural network represent your data?
  - Can the optimization algorithm find that good representation?
- More on efficiency this afternoon…
- Does that representation generalize?

Two rules!

Rule #1: MUST hold out some data and check error *at very end*

Rule #2: DO NOT extrapolate to inputs outside literal training domain

mathematical sense of word

Thank You!
Any questions?

Thinking ahead to this afternoon: how would you parallelize gradient descent?